PAPER CHECK SYSTEM

Yufan Cai
516030910496
Shanghai Jiao Tong University
gluttony@sjtu.edu.cn
Group Leader

Xun Xu
516030910516
Shanghai Jiao Tong University
459657718@qq.com
Group Member

Chao Jin
516030910476
Shanghai Jiao Tong University
827812522@qq.com
Group Member

June 16, 2019

1 Introduction

Author: Chao Jin

In recent years, academic misconduct like plagiarism has occurred frequently in domestic academic circles. Nowadays, major universities have become the most serious areas of plagiarism. There are not only a large number of plagiarized papers, but also a high probability of occurrence. In addition to undergraduate students who plagiarize when writing papers, academic misconduct of professors, lecturers, masters and doctors seems to be unable to stop. In order to deal with this situation, most universities have taken some effective measures. The most used means by universities is to use technical means to conduct paper check.

First of all, paper check is to clearly demonstrate the school's determination to severely crack down on such academic misconduct and academic fraud, so that tutors and students will get aware of the seriousness of paper plagiarism and academic misconduct and establish a correct attitude towards paper writing. Paper check plays a correct guiding role in paying attention to the originality of the paper and the emphasis on the citation. Furthermore, without paper check system, paper check only relies on the personal judgment of tutors and the students. However, faced with massive information in the Internet era, personal judgment inevitably have great limitations. Paper check system can also help them avoid falling into the risk of academic misconduct. More importantly, from the perspective of academic moral education, paper check is a form of academic criterion education. Rigorous detection of student paper repetition rate starting from undergraduate studies makes them realize the importance of academic criterion and academic ethics and raise their awareness of intellectual property from the outset. This plays a positive role in ensuring the healthy development of academic research in China in the future.

In our project, we proposed our own paper check system. In the following sections, we will introduce related work we knew about, difficulties we faced, innovations we did and complete system we designed. Finally, we will plan future work about our project.

2 Related Work

Author: Xun Xu

In Text Similarity Computing Based on Context Framework Model, Jin Yaoyong, it introduce an algorithm based on context framework model which contains three factors, field, situation and background. Field is the range that the event belongs to and situation refers to the dynamic description of the event and background centers on the specific background type of the event.

In Research on Chinese Document Copy Detection Based on Extracting Key Words, MA hui-dong, LIU Guo-hua,LI Xu, LIANG Peng, LIU Chun-hui, ZHANG Ling-yu, it draws lessons from vector space model. According to feature vectors consisting of words frequency, it use dot product and cosine formula to compute the similarity between papers.

In Determining and characterizing the reused text for plagiarism detection, Fernando Sánchez-Vega,Esaú Villatoro-Tello,Manuel Montes-y-Gómez, Luis Villaseñor-Pineda,Paolo Rosso, it comes up with a method called the Rewriting Index method, which is able to identify portions of reused text even if they have suffered from some modifications.

3 Task Division

Yufan Cai: Check algorithm and similarity detecting system design.

Xun Xu: Database Connection.

Chao Jin: PDF Extraction and search engine system design.

4 Difficulties and Innovations

Author: Yufan Cai

We are confronted with a engineering problem, so we more focused on the practical problem such as how to run the similarity-detection system as fast as possible within a million papers database. The specific problems and our innovation solutions are specified below.

4.1 Efficiency: Six-tuple Factorization

The first problem is that there are too many papers in the database. If we check one paper with all the other papers, it will cost a lot of time even lasting for two days. On the other hand, Most papers are completely different because they study different fields, so it is no needs to compare a paper with all the other paper together. Our idea is to figure out which paper is more probably similar to the paper we want to check. We integrate the information we have now, and use a six-tuple for preprocessing.

Consider a paper with author, title, year, keyword, abstract, text. We first extract the paper into six-tuple, and each element is as follows: author, year, title, keyword, abstract, text. Then, we first check the author name, and check the year. If they are same, we can't say that they are plagiarism. if not, then we check the title and the keywords, because they can reflect the center of the article and the main-point of the research. If the two passage have two or more same keywords, then we further check the abstract and the main text. This method can filter about 95% of the paper in the database and 5% related paper for further detection. In the further experiment, we can find that this six-tuple method can find the related paper precisely, and the detecting result is the same as the original method without preprocessing filter.

4.2 Precision: Metric Algorithm

Another difficulty is that how to measure the similarity of two papers. Text similarity check is mainly used in Re-check detection of Papers, the deduplication of search engines and other fields. The traditional methods are to measure twelve consecutive words in a sentence. If the two sentence has at least twelve consecutive words, we assert that they are similar. However, it's extremely fussy to extract feature items with the traditional methods for computing the text similarity. In addition, it will bring uncertainty to select elements randomly. For example, if someone change just one word or insert one word in a sentence, the algorithm will not judge the similarity. We check many inference, and conclude as follows.

There are a wide range of methods for calculating the similarity in meaning between two sentences. Here we take a look at the most common ones. The easiest way of estimating the semantic similarity between a pair of sentences is by taking the average of the word embeddings of all words in the two sentences, and calculating the cosine between the resulting embeddings. Obviously, this simple baseline leaves considerable room for variation. We'll investigate the effects of ignoring stopwords and computing an average weighted by tf-idf in particular. One interesting alternative to

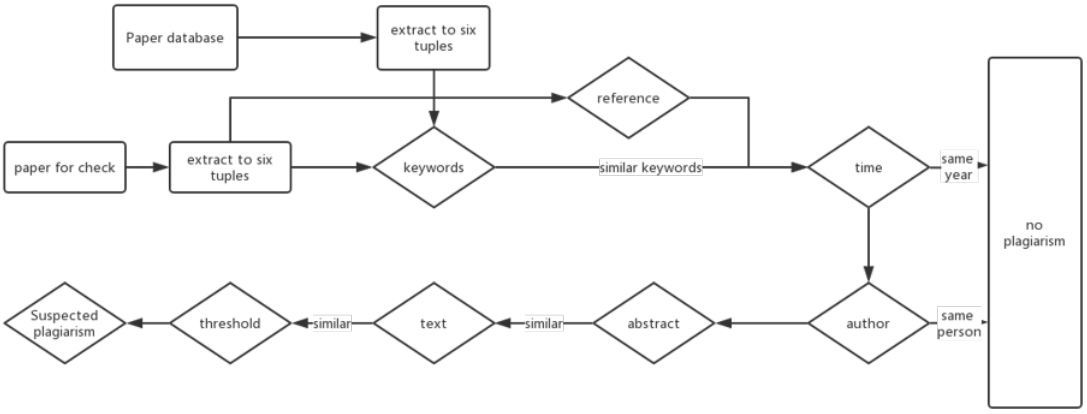


Figure 1: Six-tuple Extraction

our baseline is Word Mover's Distance. WMD uses the word embeddings of the words in two texts to measure the minimum distance that the words in one text need to "travel" in semantic space to reach the words in the other text.

Another text similarity method based on improved Jaccard coefficient is proposed. This method takes into account the weights of elements and samples in the document, even the contribution degree to multiple text similarity. The results suggest that the text similarity method based on the improved Jaccard coefficient has been proved to be effective with a satisfactory accuracy, which can be applicable to various lengths of Chinese, English documents. It effectively solves the problem of inexact computing with existing technologies.

4.3 Practical: Parallel and Search Engine

In practical system, we implemented two methods. On one hand, we use six-tuple extraction and Jaccard coefficient similarity method to check the papers. In the meantime, we use parallel process to speed up the check algorithm. In practise, we use thirty-two cpu kernels and it will reduce the check time about ninety-five percent. On the other hand, we implement search engine method. We use inverted index algorithm to build the index for paper database. For every check, we search the sentence in the index and get the information of the similar sentence, which includes the paper's title and author. Then, for specify check, we use the suspected paper's title and author in the check algorithm for scrutinizing the plagiarism degree. More details can be seen in Section 5.5.1.

4.4 PDF Extraction: Grobid

Author: Chao Jin

To check papers, we need TXT files containing each paper's content rather than PDF files. So we have to convert source PDF files to TXT files. We tried two different methods and eventually decided to use *grobid* to extract PDF files. More details will be introduced in Section 5.3.

5 Proposed System

5.1 Environment

Python2.7 For search engine system design.

Python3.5 for the other parts.

5.2 Database Connection

Author:Xun Xu

To fetch the Paper IDs of the papers related to the paper to be checked, we have to perform database connection. There are three tools I use to make the task easier.

First, I use WinSCP to check the server, where I found papers are saved in folders with the filenames containing its own Paper ID in the form of PDF. Then I use Navicat to make the database visualized. I found every paper is saved with its information, such as title, time, authors, etc. Paper ID is not saved directly but need to be segmented. To confirm that I fetch the correct PDF file from the server according to the Paper ID gained in the database, I use Xshell5 to download the PDF file to local and check whether it is the matched file, for the number of PDF files in a folder is too large to open the folder through WinSCP.

In detail, I use pymysql to perform the database connection. I store the information in a dictionary. The dictionary has two layers.The first key is Paper ID and the second keys are authors,title,abstract and etc. The information stored can be used in the search engine. The framework of database connection is shown as Fig 2.

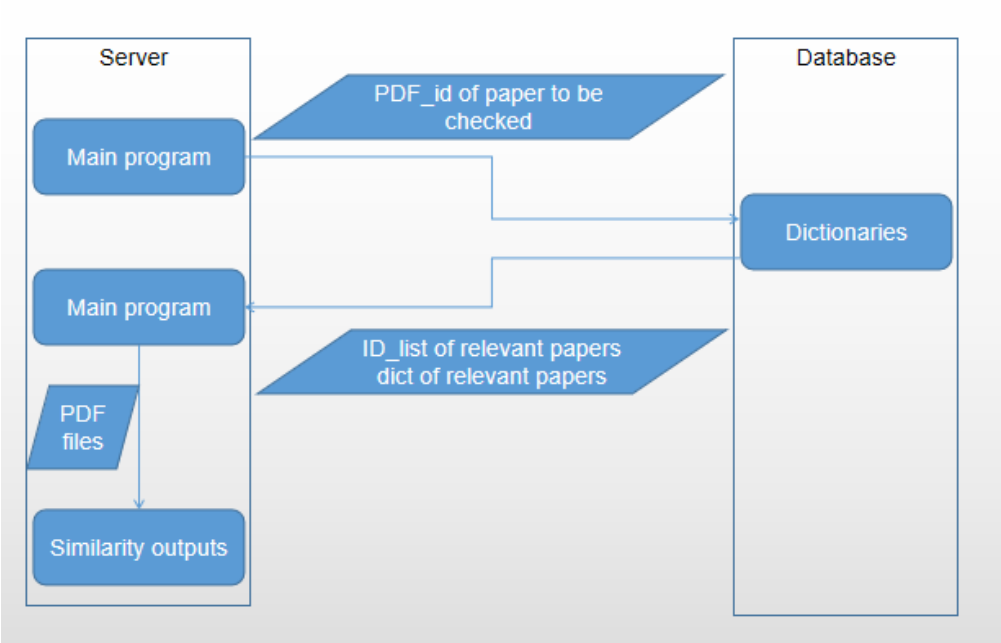


Figure 2: Framework of Database Connection

5.3 PDF Extraction

Author: Chao Jin

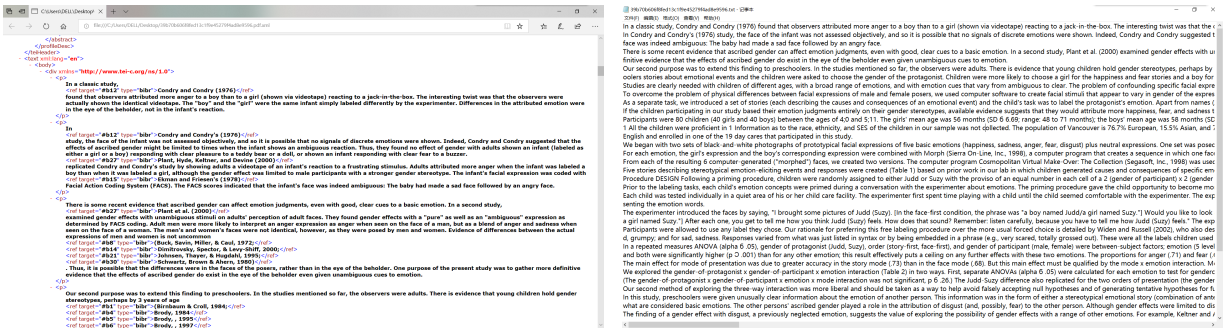
To check the similarity of two papers, first we should do is to extract their content. Now that some messages of each paper such as title, authors, date, keywords, etc are already stored in the database, we just extracted the body part of each paper.

First, we used the python package *pdfminer*. To our disappointment, its effect was not good because of its bad adaptability to papers in different formats. Then, we tried *grobid*. By using *grobid*, we converted PDF files to TXT files via XML files. Through observing those XML files, we found a fact that the body part of a paper is contained in the label <*body*> and one label <*p*> represents one paragraph. The detailed process to extract PDF files is as follows:

1. Install the latest version of grobid.
2. Use the command line to enter the installation directory and use the command *./gradlew run*.

3. In Python, pass the PDF files to be converted to *http://localhost:8070/api/processFulltextDocument* by using method requests.post() and save corresponding XML files. A sample XML file converted from a PDF file is shown as Fig 3(a).

4. Read and analyze XML files, obtain all contents of all labels <*p*> in the label <*body*> and save them in corresponding TXT files. A sample TXT file converted from a XML file is shown as Fig 3(b).



(a) A Sample XML File (b) A Sample TXT File

Figure 3: Sample Files

By the way, other messages such as title, author, date, abstract, keywords and references are also contained in different labels of XML files, which can be obtained in a similar way if necessary.

5.4 Check Algorithm

Author: Yufan Cai

5.4.1 Main Steps

1. Given the parameter K, K is the size of the moving window in the document. Given two documents X and document Y with lengths n_1 and n_2 respectively, determine the number of elements in the document with length K, and calculate each The proportion of elements in the document;

2. Calculate the similarity of each element;

3. Calculate the proportion of each element in all elements of length K;

4. Determine the weight of each K-word element;

5. Summarize the similarity of all K-word elements and calculate the document similarity.

5.4.2 More Details

1. Determine the number of elements in the document with a length of K and calculate the proportion of each element in the document. Assuming that the document lengths of documents X and Y are n_1 and n_2 respectively, document X contains n_1 elements of length 1, containing $n_1 - 1$ elements of length 2, and so on, document X contains 1 elements of length n_1. The size of the sliding window of these elements is 1. The sliding window slides from the starting position of the text to the ending position to form n-Gram. So it contains $n_1 - K + 1$ elements with length is equal to K in the document X. Similarly, it contains $n_2 - K + 1$ elements with length is equal to K in the document Y. The weight for every K elements is as follows:

$$NX_w = \frac{|X_w|}{n_1 - K + 1}$$

$$NY_w = \frac{|Y_w|}{n_2 - K + 1}$$

2. Compute the similarity. We define the similarity between document X and document Y is the ratio of the intersection and the union result. If element w_i exists in both X and Y, then this element correspond to the similarity is as follows:

$$C(X_{w_i}, Y_{w_i}) = \frac{|X_{w_i} \cap Y_{w_i}|}{|X_{w_i} \cup Y_{w_i}|} = \frac{min(|X_{w_i}|, |Y_{w_i}|)}{max(|X_{w_i}|, |Y_{w_i}|)} = \frac{min(NX_{w_i}, NY_{w_i})}{max(NX_{w_i}, NY_{w_i})}$$

3. Calculates the proportion of each element in all elements of length K. Use $\phi(w_i)$ to represent the weight of element w_i all n-Gram element with length K in document X and Y, then we can compute as follows:

$$\phi(w_i) = \frac{|X_{w_i}| + |Y_{w_i}|}{n_1 - K + 1 + n_2 - K + 1}$$

4. Determine whether the element has any contribution to similarity. Use $F(w_i)$ to represent the appearance of element w_i, then we can define: if $w_i \in X \cap Y$, then $F(w_i) = 1$; if $w_i \in X \cup Y$, $F(w_i) = 0$.

5. Calculate the document similarity.

$$Similarity(X, Y) = \frac{\sum_{w_i \in \sum K} C(X_{w_i}, Y_{w_i})\phi(w_i)}{\sum_{w_i \in \sum K} F(w_i)\phi(w_i)}$$

5.5 System Design

5.5.1 Search Engine System

Author: Chao Jin

We used library *Lucene* to achieve our search engine. *Lucene* is an efficient, Java-based full-text search (The process of first indexing and then searching the index is called full-text search) library. Full-text search is roughly divided into two processes, *Indexing* and *Search*. The framework of our search engine is shown as Fig 4.

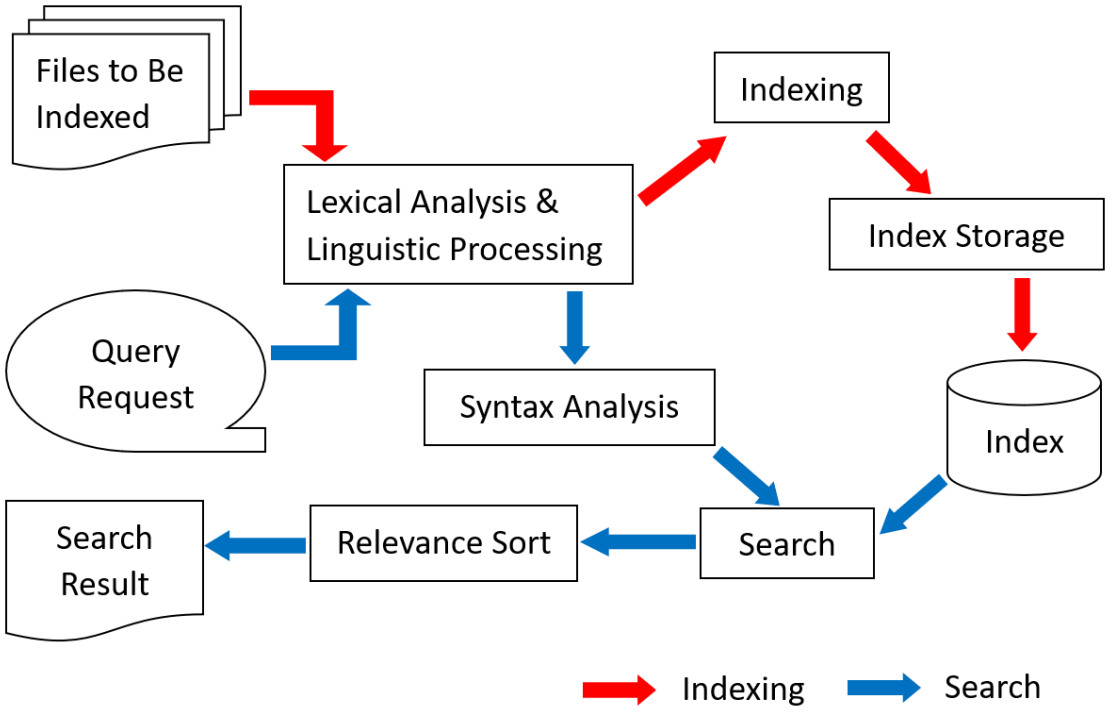


Figure 4: Search Engine Framework

Indexing: Create indices by extracting information from all data in the real world

1. Some documents to be indexed. For example, there are two documents as follows.
   Document One: Students should be allowed to go out with their friends, but not allowed to drink beer.
   Document Two: My friend Jerry went to school to see his students but found them drunk which is not allowed.
2. Pass the original documents into the tokenizer. The tokenizer divides the documents into single words, removes punctuation and stop words. Tokens obtained from the two documents are as follows: *Students*, *allowed*, *go*, *their*, *friends*, *allowed*, *drink*, *beer*, *My*, *friend*, *Jerry*, *went*, *school*, *see*, *his*, *students*, *found*, *them*, *drunk*, *allowed*.
3. Pass tokens to the linguistic processor. The linguistic processor converts tokens to lowercase, gets stemming like converting *cars* to *car* and does lemmatization like converting *drove* to *drive*. Terms obtained from the above steps are as follows: *allow*, *go*, *their*, *friend*, *allow*, *drink*, *beer*, *my*, *friend*, *jerry*, *go*, *school*, *see*, *his*, *student*, *find*, *them*, *drink*, *allow*.
4. Pass terms to indexer. The indexer creates a dictionary using terms, sorts the dictionary alphabetically, merges the same terms and gets a posting list shown as Fig 5. The field *Document Frequency* represents the number of documents containing corresponding terms. The fields *Frequency* represents the number of terms contained in the corresponding documents.

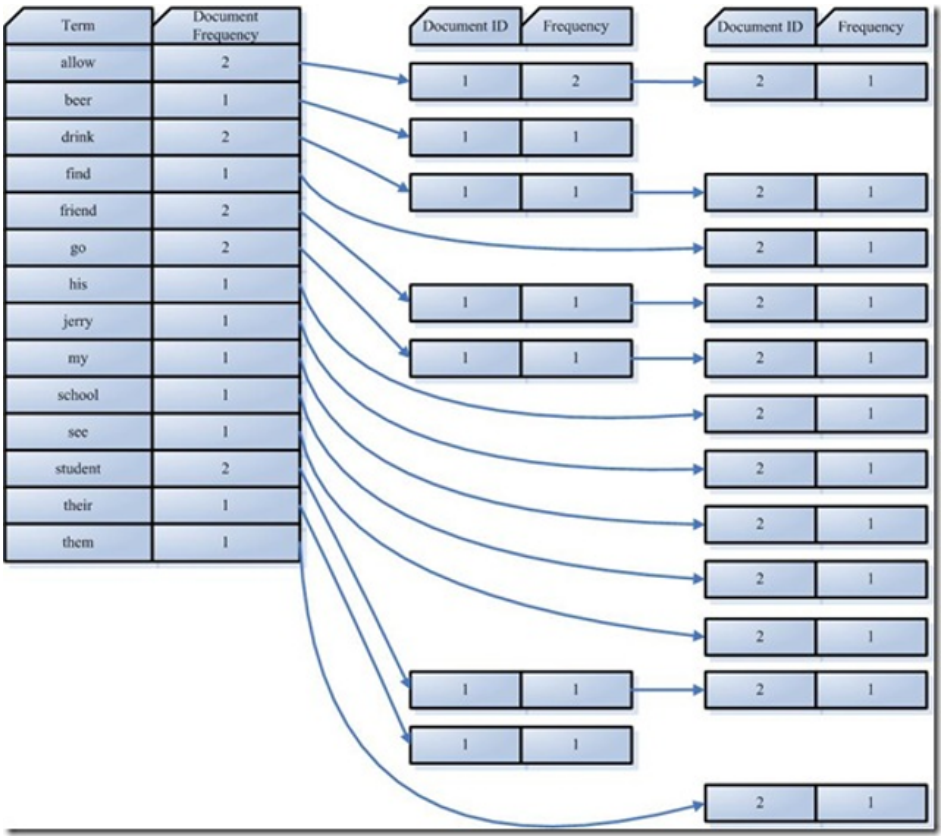


Figure 5: Posting List

Search: Get the user's query request, search for the created index, and then return the results

1. The user inputs a query request. For example, the request *lucene AND learned NOT hadoop* represents that user attempts to find documents containing *lucene* and *learned* but *hadoop*.
2. Do lexical analysis, linguistic processing and syntax analysis for the query request.

(a) Lexical analysis: Identify words and keywords. In the request above, words include *lucene*, *learned* and *hadoop*, and keywords includes *AND* and *NOT*.

(b) Linguistic processing: Same with one in *Indexing*. In this case, convert *learned* to *learn*.

(c) Syntax analysis: As is shown as Fig 6, form a syntax tree according to the syntax rules of the query request.

3. Search indices and get documents matching the syntax tree.
4. Sort the results based on the relevance of the matching documents and the query request.

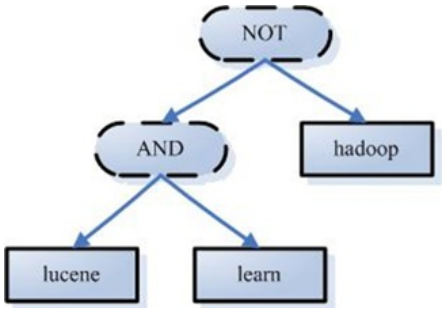


Figure 6: Syntax Tree

The sample search results are shown as in Fig 7. Now there is no titles, authors and so on in the results, but these messages can be displayed if necessary.

```
Hit enter with no input to quit.
Query:Hot Charged Rolling

Searching for: Hot Charged Rolling
50 total matching documents.
-----------------------
path: testfolder/7d00547f798283d8fcf4639dd5c8fb9e1174.txt
name: 7d00547f798283d8fcf4639dd5c8fb9e1174.txt
title:
author:
language:
-----------------------
path: testfolder/0e9e25f833dca8c58fb457b2021453d93a2b.txt
name: 0e9e25f833dca8c58fb457b2021453d93a2b.txt
title:
author:
language:
-----------------------
path: testfolder/793b59f3a9913ce2752fe33cf32c5b430e5c.txt
name: 793b59f3a9913ce2752fe33cf32c5b430e5c.txt
title:
author:
language:
```

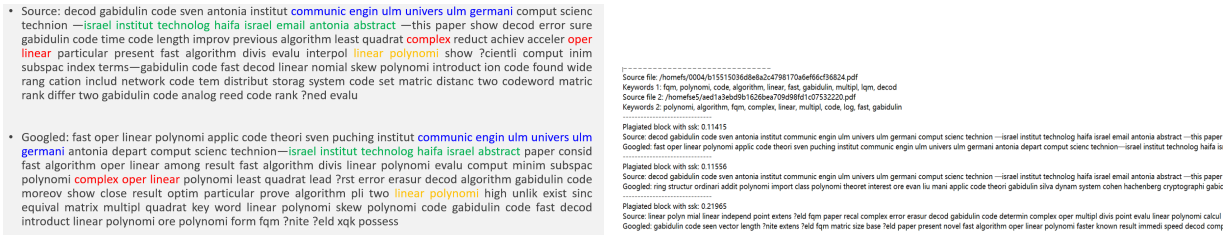
Figure 7: Sample Search Results

5.5.2 Similarity Detecting System

Author: Yufan Cai

Compared with the Search Engine System, the tradition similarity detecting system will be more complicated. Here is the check steps. We firstly get a new paper and use PDF extraction to extract it into six-tuple. Secondly, we request a comparison request to the database system, to find out the related papers. The detail steps are that we first compare the keywords and if there are more than two same keywords, then we classify these paper to suspected related

paper. Then we compare the paper's author and year, if these two are not the same, then we finally define them as related paper. Then we check the related paper using the similarity detecting algorithm including using removing stop words, stem extraction, lexical reduction and so on. The results are shown as Fig 8.



(a) A Sample Result for Compare Two Papers (b) A Check Algorithm Sample Result

Figure 8: Sample Results of Check System

6 Future Work

We will try to design a interface for our system to achieve visualization, which will make our system more convenient to use.